# Pre-Trained Denoising Autoencoders Long Short-Term Memory Networks as probabilistic Models for Estimation of Distribution Genetic Programming

Master Thesis - M.Sc. Business Education

---


Student: Roman Höhn

Date of Birth: 1991-04-14

Place of Birth: Wiesbaden, Hesse

Student ID: 2712497

Supervisor: David Wittenberg


---

Master Thesis


FB 03: Chair of Business Administration and Computer Science

Johannes Gutenberg University Mainz



Date of Submission: 2023-01-15

# Contents

## List of Tables



## List of Figures

# 1 Abstract

...

# 2 Introduction

Denoising Autoencoder Genetic Programming (DAE-GP) is a novel variation of an genetic programming based Estimation of Distribution Algorithm (EDA-GP) that uses a denoising autoencoders long short-term memory network (DAE-LSTMs) as a probabilistic model to sample new candidate solutions [18].

DAE-LSTMs are recurrent neural networks (RNNs) that can be trained in an unsupervised learning environment to minimize a reconstruction error for encoding input data into a compressed representation and subsequently decoding the compressed representation back to the input dimension. In DAE-GP, DAE-LSTMs are trained with a subset of high-fitness solutions selected from a parent population with the aim to capture their promising qualities. The resulting model is then used to sample new offspring solutions by propagating partially mutated solutions from the parent population through the DAE-LSTMS [18]. In previous work DAE-GP has been shown to outperform GP for both a generalized version of the royal tree problem [18] as well as for a real-world symbolic regression problem [17].

The DAE-GP algorithm first described by [18] trains a DAE-LSTMs for each generation $g$ of the search from scratch. [17] and [16] suggests the incorporation of a pre-training strategy into the evolutionary search as a possible way of improving the model quality of DAE-GP. The key idea is to pre-train an initial DAE-LSTM on a large population of candidate solutions and to use the pre-trained parameters of this initial model in each generation of the search as starting parameters for the current generation DAE-LSTM. This thesis studies the influence of using pre-trained DAE-LSTMs in DAE-GP for symbolic regression, especially looking at the influence in generalization behavior and the overall quality of solutions found by DAE-GP. The aim of this study is to answer the question if a pre-training strategy can help to improve generalization in DAE-GP and if so, analyze how this improved generalization ability benefits the overall performance of DAE-GP.

...

# 3 Theoretical Foundations

*This section describes the relevant concepts that are necessary for the understanding and classification of DAE-GP as well as the concept of pre-training in artificial neural networks*

## 3.1 Denoising Autoencoder Genetic Programming

DAE-GP is an EDA-GP algorithm that uses DAE-LSTM networks as a probabilistic model to sample new offspring solutions [18].

### 3.1.1 Evolutionary Computation

As a variant of GP, DAE-GP can be classified as a meta-heuristic that belongs to the field of evolutionary computation (EC). EC based meta-heuristics are optimization methods that simulate the process of Darwinian evolution to search for high quality solutions by applying selection and

variation operators to a population of candidate solutions. Examples of EC include genetic algorithms (GA), evolutionary strategies (ES) and GP. In EC, the quality of a solution is commonly measured as fitness and the time steps of the search are called generations. Another important concept in EC is the distinction between genotypes and phenotypes of solutions, the genotype contains the information that is necessary to construct the phenotype, the outer appearance of a particular solution on which we measure the overall quality of solutions. The representation of a solution is therefore defined by the mapping of genotypes to phenotypes [12]. Genetic operators, such as mutation or recombination are usually applied to the genotype of solutions.

### 3.1.2 Genetic Programming

GP follows the same basic evolutionary principle of EC but searches for more general, hierarchical computer programs of dynamically varying size and shape [8]. The computer programs that are at the center of the evolutionary search in GP are commonly represented by tree structures at the level of their phenotype [12]. Since GP searches for high fitness computer programs that produce a desired output for some input, it can be applied to various different problem domains such as symbolic regression, automatic programming, or evolving game-playing strategies [8]. An important quality of GP is the ability to search for solutions of variable length and structure. GP is an especially useful meta heuristic for problems where no a priori knowledge about the final form of good solutions is available. GPs ability to optimize solutions for their structure as well as for their parameters led to it being one of the most prevalent methods used for symbolic regression [10].

### 3.1.3 Estimation of Distribution Algorithms

The aim of Estimation of distribution algorithms (EDA) is to replace the standard variation operators used in GA by building probabilistic models that can capture complex dependencies between different decision variables of an optimization problem [12]. EDAs use this probabilistic model to sample new offspring solutions inside an evolutionary search to replace crossover and/or mutation operators.

### 3.1.4 Denoising Autoencoders

One possible way of model building in EDA proposed by [11] is to use denoising autoencoders (DAEs) as generative models to capture complex probability distributions of decision variables. DAE, a variation of the autoencoders (AE), are a widely used type of neural networks in the domain of unsupervized machine learning that maps $n$ input variables to $n$ output variables using a hidden representation.

AE were introduced by [6] to compress high-dimensional data into lower-dimensions. An AE consists of two different subunits:

- Encoder $g(x)$: Encode input data to a smaller cental layer $h$
- Decoder $d(h)$: Decode and output the the encoded data back to its original dimension

The AE is trained to reduce the reconstruction error between input and output data, after the training procedure is finished the network is able to reduce the dimensionality of input data to a compressed representation[6].

DAE was first introduced by [14] as an improved AE with the ability to learn new representations of data that is especially robust to partially corrupted input data. DAE modifies the AE by using

partially corrupted input data for the AE and training it to reconstruct the uncorrupted, original version of the input data.

Since the hidden representation of DAE captures the dependency structure of the input data it can therefore also be used to generate new solutions in the context of GAs [11].

DAE-GP builds upon the concept of using DAEs in EDAs described by [11] and transfers the concept to the domain of GP. The mutation and crossover operators of standard GP are replaced by sampling new solutions from a probabilistic model that is build by training a DAEs long short-term memory (LSTM) network on a subset of high fitness solutions from the current population [18].

LSTMs are a variant of recurrent neural networks first introduced by [7] that can store learned information over en extended time period while avoiding the problem of vanishing and/or exploding gradients. Since DAE-GP encodes candidate solutions as linear strings in prefix notation, the DAE in DAE-GP uses LSTMs for both encoding and decoding where the total amount of time steps $T$ is equal to sum of the length of the input solution and the output solution [18].

### 3.2 Pre-Training

Pre-Training describes the concept of initially training a neural network on a large dataset before using it to solve a more specific task. It is a commonly used strategy in deep architectures that has been shown to improve both the optimization process itself as well as the generalization behavior if compared to the standard approach of using randomly initialized parameters [3].

The strategy of pre-training artificial neural networks is based on the idea of transfer learning that aims at retaining previously learned knowledge from one task to re-use it for another task [5] [9]. Transfer learning traditionally follows a two phase approach:

1. Pre-Training Phase: Capture knowledge from source task
2. Fine-Tuning Phase: Transfer knowledge to the target task

where source task and target task are usually similar but may differ in their feature space and the distribution of training data [9].

Some of the main motivations for using pre-training is the ability to reduce the need for large amounts of training data which can often be unavailable or too expensive to collect as well as an improved performance by either reducing the computational effort that is necessary to train a model to solve a specific task or by improving the models generalisation ability.

. . . describe different pre-training strategies, e.g. few shot, perpetual, re-using. . .

– describe unsupervized vs supervized pre-training. . .

## 4 Pre-Training in DAE-GP

The pre-training strategy used in this thesis is applied to the DAE-LSTM model $M_g$ that are used in each DAE-GP generation $g$ for $g \in 1, ..., g_{max}$. Instead of initializing and optimizing each model from scratch as done in previous work (e.g. [18] [17]), a separate DAE-LSTM network $\hat{M}$ will be trained on a large population of randomly initialized solutions.
The trainable parameters $\theta_{\hat{M}}$ obtained after finishing the training procedure of $\hat{M}$ are then used as the starting parameters for each following DAE-LSTM model $M_g$ for $g \in 1, ..., g_{max}$.

The motivation for incorporating pre-training into DAE-GP is based on the following suspected mechanisms of improvement:

1. Improve overall performance of DAE-GP by improving either run time and/or solution quality
2. Reduce the need for large population sizes to avoid sampling error
3. Improve model robustness and generalization ability

The probably most obvious motivation for using a pre-training strategy in DAE-GP is based on the fact, that initializing $M_g$ with pre-trained weights $\theta_{\hat{M}}$ is likely to reduce the amount of training epochs that are necessary until the point of training error convergence is reached.

Early research on pre-training for DAE by [3] comes to the conclusion that besides adding robustness to models the strategy also results in improved generalization and better performing models. The authors describe that unsupervised pre-training behaves like a regulizer on DAE networks, the mean test error and it's variance are reduced for DAE networks that are initialized from pre-trained weights if compared to the same architectures that use random initialization.

Another important finding by [3] is that the positive effect of pre-training described above is dependent on both the depth of the network as well as the size of layers - while increasingly larger networks benefit increasingly more from pre-training the final performance of small architectures tends to be worse with pre-training than with randomized weight initialization. The authors find evidence that pre-training DAE is especially usefull for optimizing the parameters of the lower layers of the network.

Another motivation for introducing pre-training into DAE-GP is the prevalence of sampling error for small GP populations sizes. [13] finds that sampling error, non-systematic errors that are caused by observing only a small subset of the statistical population, is a severe problem in the domain of GP since the initial population may not be a representative sample of all possible solutions. [13] also introduces a method for calculating optimal population sizes to minimize the presence of sampling error. Since DAE-LSTM of DAE-GP learns the properties of it's training population and reuses the acquired knowledge in the sampling procedure, using the parameters obtained from training $\hat{M}$ on a sufficiently large training population might reduce the need for large population sizes in the following generations of the search if $\hat{M}$ already implicitly captured the properties of a representative sample of solutions. If this mechanism can be applied successfully it would benefit the performance of DAE-GP by increasing the population diversity (resulting in better solution quality) as well as by reducing the need for large population sizes which require computational resources.

### 4.1 Implementation

The DAE-GP algorithm, first described by [18], is summarized as a flowchart in figure §. After setting the generation counter $g$ to 0 and creating the initial population of solutions $P_0$, DAE-GP enters the main loop that checks if a termination criteria is satisfied (i.e. maximum number of generations or fitness evaluations is reached). For each generation $g$ the fitness of all new individuals in $P_g$ is evaluated. During model building, DAE-GP selects a subset $X_g$ of the current population that is used as training data for the DAE-LSTM model $M_g$ which is trained to learn the properties of $X_g$. The trained model $M_g$ is then used for model sampling, partially mutated solutions from $P_G$ are propagated through $M_g$ to create a new population $P_{g+1}$. This process is repeated until the termination criteria is met and DAE-GP returns the highest fitness solutions.

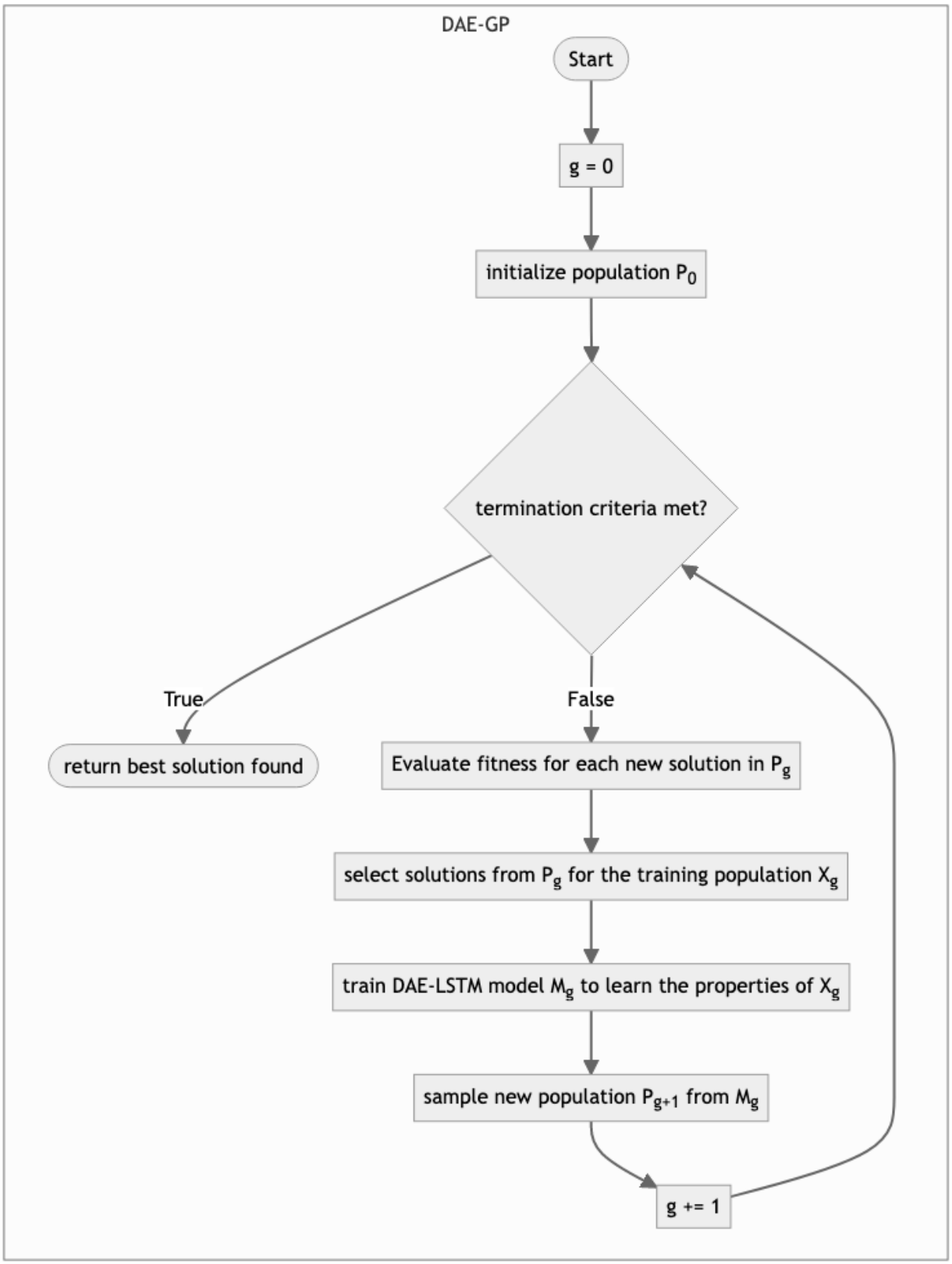



Figure 1: Flowchart - Regular DAE-GP

The pre-training strategy implemented for all experiments in this thesis is visualized as another flowchart in figure §. The main difference to regular DAE-GP is the inclusion of an initial pre-training phase where a separate population $\hat{P}$ is first initialized and then randomly split by half into $\hat{P}_{train}$ and $\hat{P}_{test}$. A DAE-LSTM model $\hat{M}$ is then trained to learn the properties of $\hat{P}_{train}$ using an early stopping training mode where we stop training as soon as the validation error for $\hat{P}_{test}$ converges. After the training for $\hat{M}$ is stopped, the current state of $\hat{M}$ is frozen and the optimized trainable parameters $\theta_{\hat{M}}$ are saved before terminating the pre-training phase to start the DAE-GP phase. During the DAE-GP phase, the only difference to the traditional DAE-GP algorithm described in figure § is during model building: We initialize $M_g$ with $\theta_{\hat{M}}$ before training it to learn the properties of $X_g$.

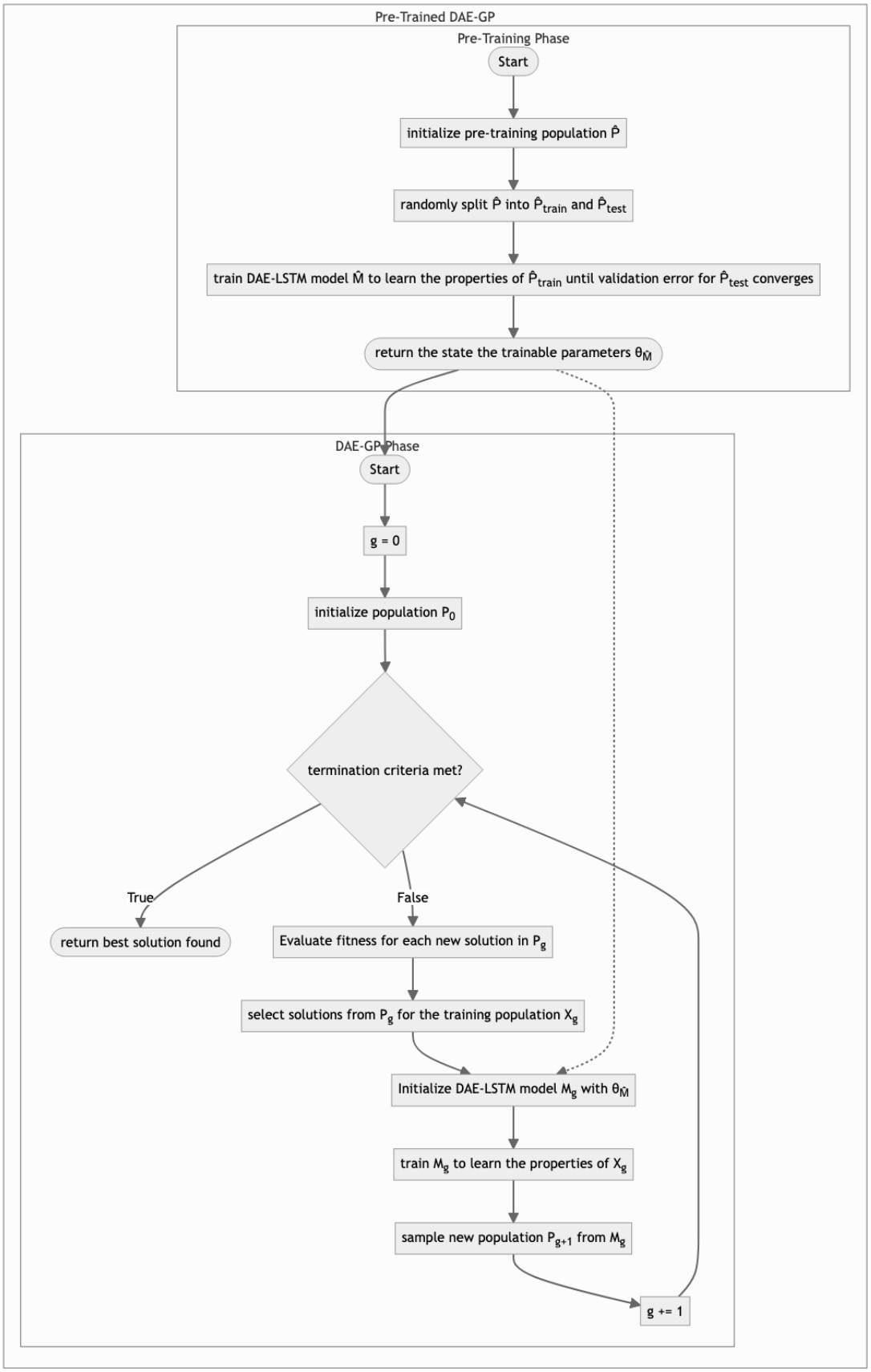



Figure 2: Flowchart - Pre-Trained DAE-GP

One difficulty in the implementation of a pre-training strategy into DAE-GP has been the detemination of the number of hidden neurons in side the DAE-LSTMs hidden layers. The original description of DAE-GP [18] used a strategy where the number of hidden neurons for each hidden layer is dynamically set per generation to the maximum individual size inside the current population. For a pre-training implementation, this strategy can not be easily adapted since it leads to a changing number of neurons at each generation resulting in different dimensions of the DAE-LSTM. To allow for the sharing of weights and biases from the pre-trained model $\hat{M}$ to each $M_g$ for $g \in 1, ..., g_{max}$ I experimented with two different strategies:

1. Set the number of hidden neurons per hidden layer for each generation to the maximum individual size inside the pre-training population
2. (dependent on the terminal/function set)

The framework that was used to conduct all experiments of this thesis was provided by the supervisor David Wittenberg and uses the python programming language in conjunction with the baseline libraries `Keras` [1] for deep neural networks and `deap` [4] for evolutionary computation.

### 4.2 Benchmark Problem

To test pre-training in DAE-GP this thesis focuses on the domain of real-world symbolic regression problems. Symbolic Regression problems have been one of the first GP applications [8] and are an actively studied and highly relevant research area. The goal in symbolic regression is to find a mathematical model for a given set of data points [10], in real-world symbolic regression these data points are sourced from real-world observations which in contrast to synthetic symbolic regression problem are more likely to contain random noise and bias. Another important challenge in solving real-world symbolic regression problems is the ability for a given model to generalize, we want the final model to show high accuracy in predicting outcomes for previously unseen cases.

The main experiments conducted in this thesis uses the NASA Airfoil Self-Noise Data Set which is part of the UCI machine learning repository [2]. The dataset consists of 5 input variables and 1 output variable that are listed in table 1.

Table 1: Airfoil - Dataset Description

| **Type** | **Name** | **Description** | **Unit** |
|---|---|---|---|
| input | x1 | Frequency | Hertz |
| input | x2 | Angle of attack | Degree |
| input | x3 | Chord length | meters |
| input | x4 | Free-stream velocity | meters/second |
| input | x5 | Suction side displacement thickness | meters |
| output | y | Scaled sound pressure level | decibels |

The objective of the airfoil problem is to find a function that accurately predicts the output variable $y$ by taking in a subset of the input variables $x1, x2, x3, x4, x5$. The function set used by all DAE-GP variations for the airfoil problem is summarized in table 2, the terminal set consists of the 5 input variables $x1, x2, x3, x4, x5$ and ephemeral random integers in the range of $[-5, .., 5]$ [17].

Table 2: Airfoil - Function Set

| **function.** | **arity** |
|---|---|
| addition | 2 |
| subtraction | 2 |
| multiplication | 2 |
| analytic_quotient | 2 |

# 5 Results

## 5.1 Effect on Generalisation

To gather a deeper understanding about the effect of pre-training DAE-LSTM in DAE-GP a series of experiments was conducted using the airfoil dataset for symbolic regression while using and different parameter configurations for the number of hidden layers as well as the number of hidden neurons per hidden layer.

To study generalization behavior this series of experiments is conducted using DAE-GP with only a single generation until the search terminates. I disregard the fitness of solutions found and focus soley on the reconstruction error that is produced during the training of each the DAE-LSTM. The reconstruction error is measured for two separate populations, a training population $P_{train}$ that is used to train our DAE-LSTM as well as a hold-out validation population $P_{test}$. For the pre-trained DAE-GP two additional, separate populations $\hat{P}_{train}$ and $\hat{P}_{test}$ are used exclusively for pre-training.

DAE-GP is tested in two different configurations:

- Variable number of hidden neurons (50, 100, 200) with a single hidden layer
- Fixed number of hidden neurons (100) per hidden layer with variable number of hidden layers (1, 2, 3)

For each configuration I tested traditional DAE-GP as well as a pre-trained DAE-GP resulting in 12 total sub experiments that were each based on 10 individual runs (total number of runs=120). To avoid creating biased results through the presence of sampling error inside the pre-training population (see [13]), the population size for the pre-training phase is chosen very high with the size of $\hat{P}$ =20000 where 50% of $\hat{P}$ is used for the training population $\hat{P}_{train}$ and the remaining 50% are used for the hold-out validation population $\hat{P}_{test}$. The training of DAE-LSTM uses a fixed number of 1000 epochs until termination and uses Adam optimization for gradient descent. The reason for using a high amount of 1000 fixed training epochs is to deliberately force the DAE-LSTM to overfit to the training data.

In general I expect that with a growing number of model parameters (either by adding more hidden layers or more hidden neurons per layer) the DAE-LSTM will be more prone to overfit to the training population $P_{train}$ resulting in a small reconstruction error for the training population and a large one for the validation population $P_{test}$. The reason for this effect is that a larger network trained over an extended period of time (without the use of strategies like early stopping), has much more potential to learn noise from the training dataset than a smaller network, which is more likely to result in worsening performance for previously unseen cases [15].

Based on the review of [3] I also expect that pre-training will have an insignificant or even negative influence on small DAE-LSTM instances while improving the networks generalization ability with

growing size.

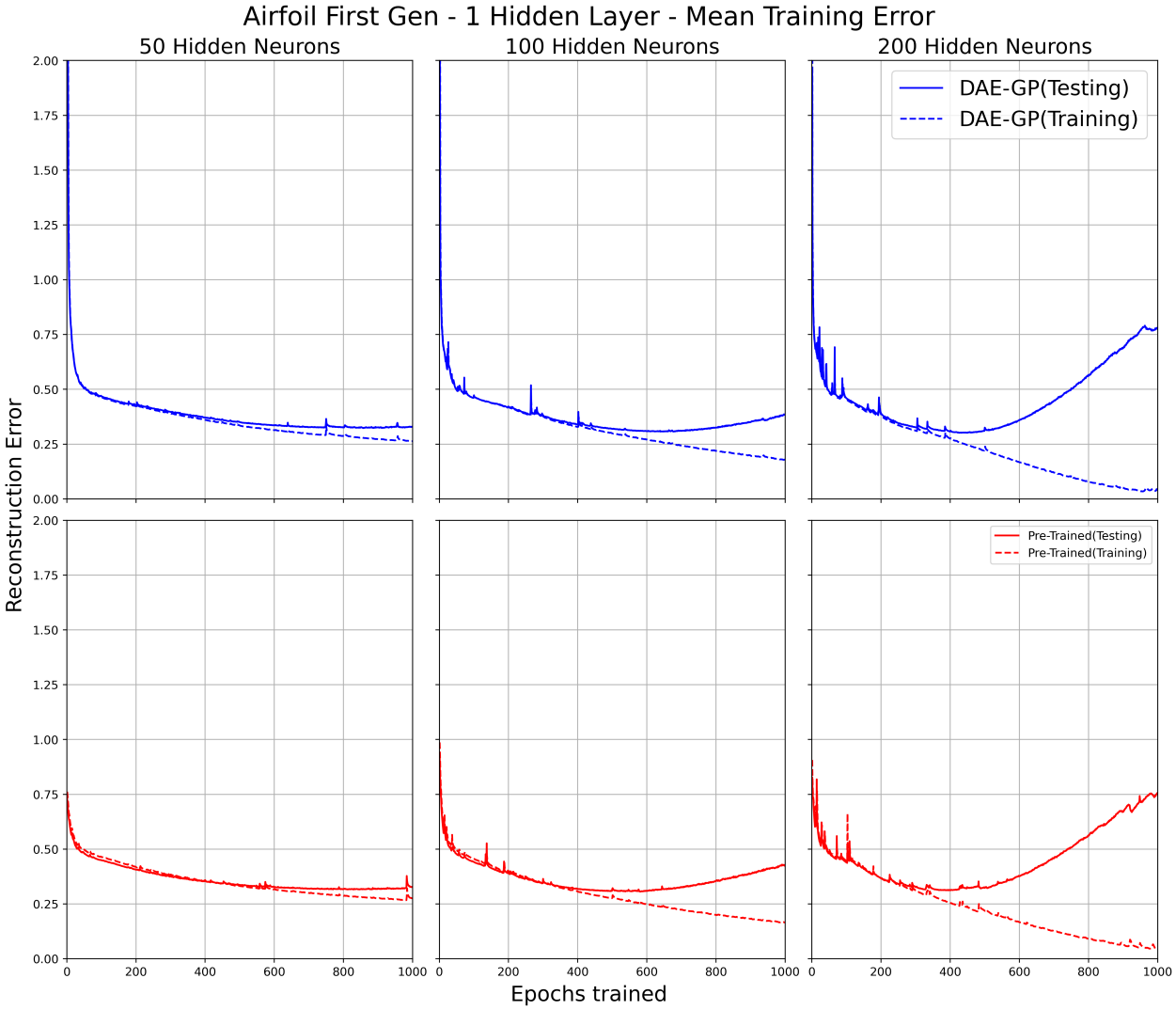



Figure 3: First Generation Median Training Error for variable number of hidden Neurons - Airfoil

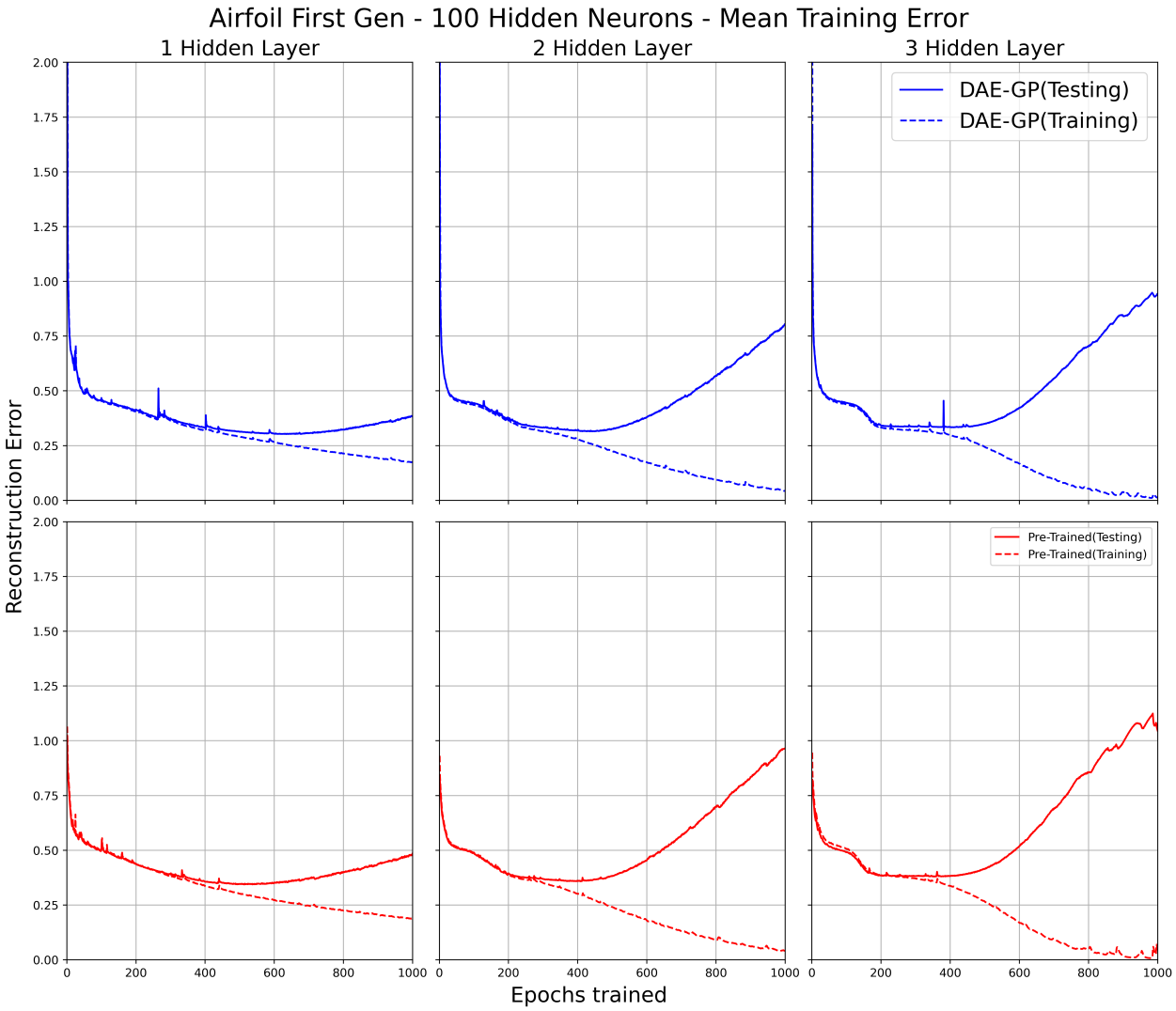



Figure 4: First Generation Median Training Error for variable number of hidden Layers - Airfoil

## 5.2 Effect on search performance

After closely examining the effect of pre-training on DAE-GPs generalization behaviour another series of experiments is conducted to study how pre-training influences the overall search behaviour of DAE-GP. Again the Airfoil problem was selected as a first benchmark problem to test pre-trained DAE-GP against traditional DAE-GP [17], the parameters are described in table *A*

Regarding the hidden neurons, the dimension of each hidden layer, I used two different strategies: Regular DAE-GP uses the same strategy as described by earlier work (see [17] or [18]), the number of hidden neurons is dynamically set at each generation to the maximum length of all solutions inside the current training population. For pre-trained DAE-GP another strategy had to be used, I used a fixed number of hidden neurons for each run that is set equal to the maximum length of all individuals that are present in the initial pre-training population. I decided to take this aproach instead of using a fixed number of hidden neurons for the fact that it simplifies the adjustment of the parameter for later experiments on different problems. It should nonetheless be mentioned that this strategy does give pre-training on average more hidden neurons per layer resulting in a more

Table 3: Airfoil - Parameter for full Run

| **parameter** | **value** |
|---|---|
| populationSize | 500 |
| generations | 30 |
| fitness | RMSE |
| TrainingMode | Convergence |
| SamplingSteps | 2 |
| hiddenLayers | 2 |
| Selection | Binary Tournament Selection |
| Pre-Training PopulationSize(Training/Validation) | 10000/100000 |
| Pre-Training TrainingMode | Early Stopping |

complex network (since the pre-training population is much larger than the regular population, the chances for larger individuals inside it are also higher).

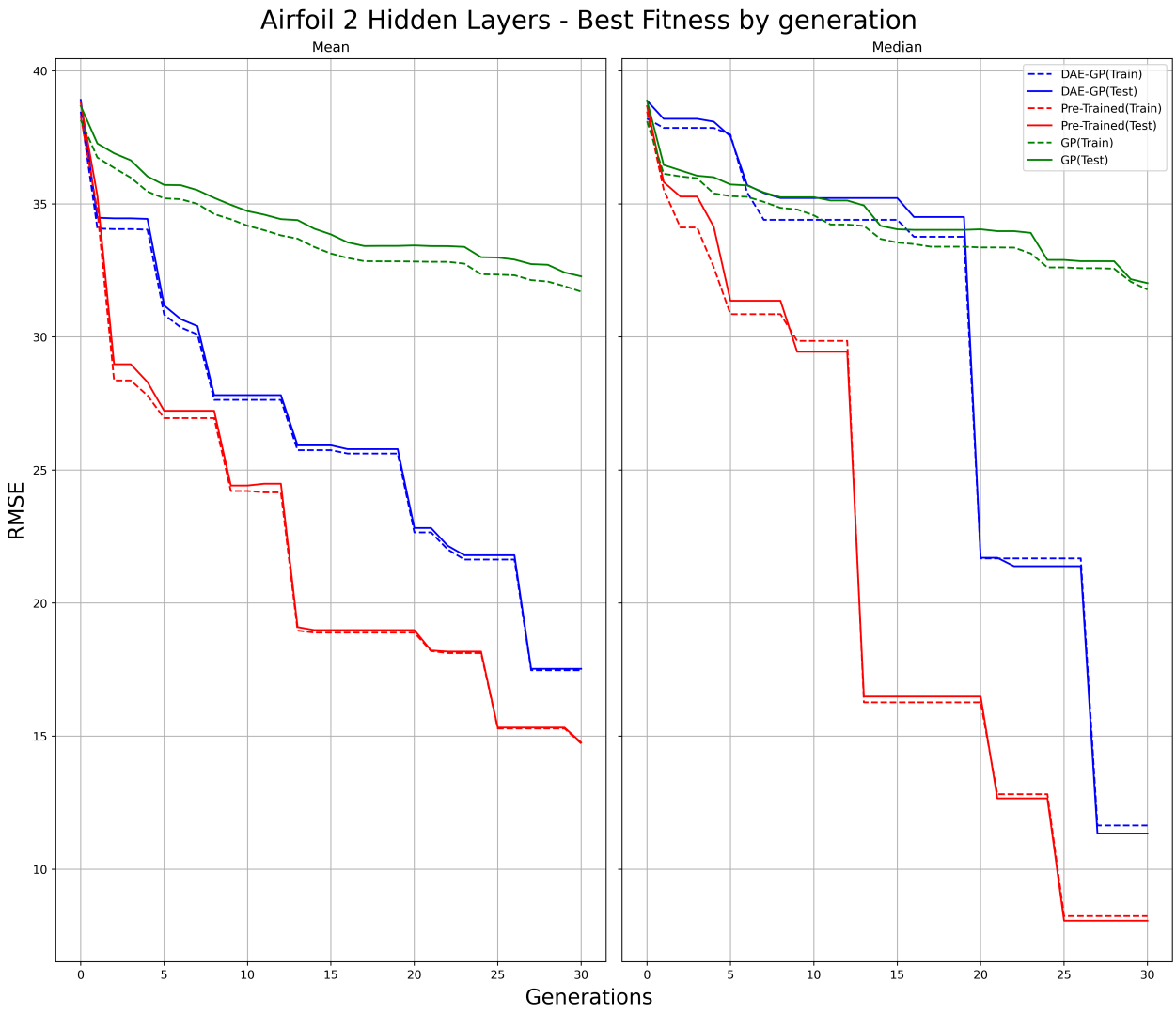
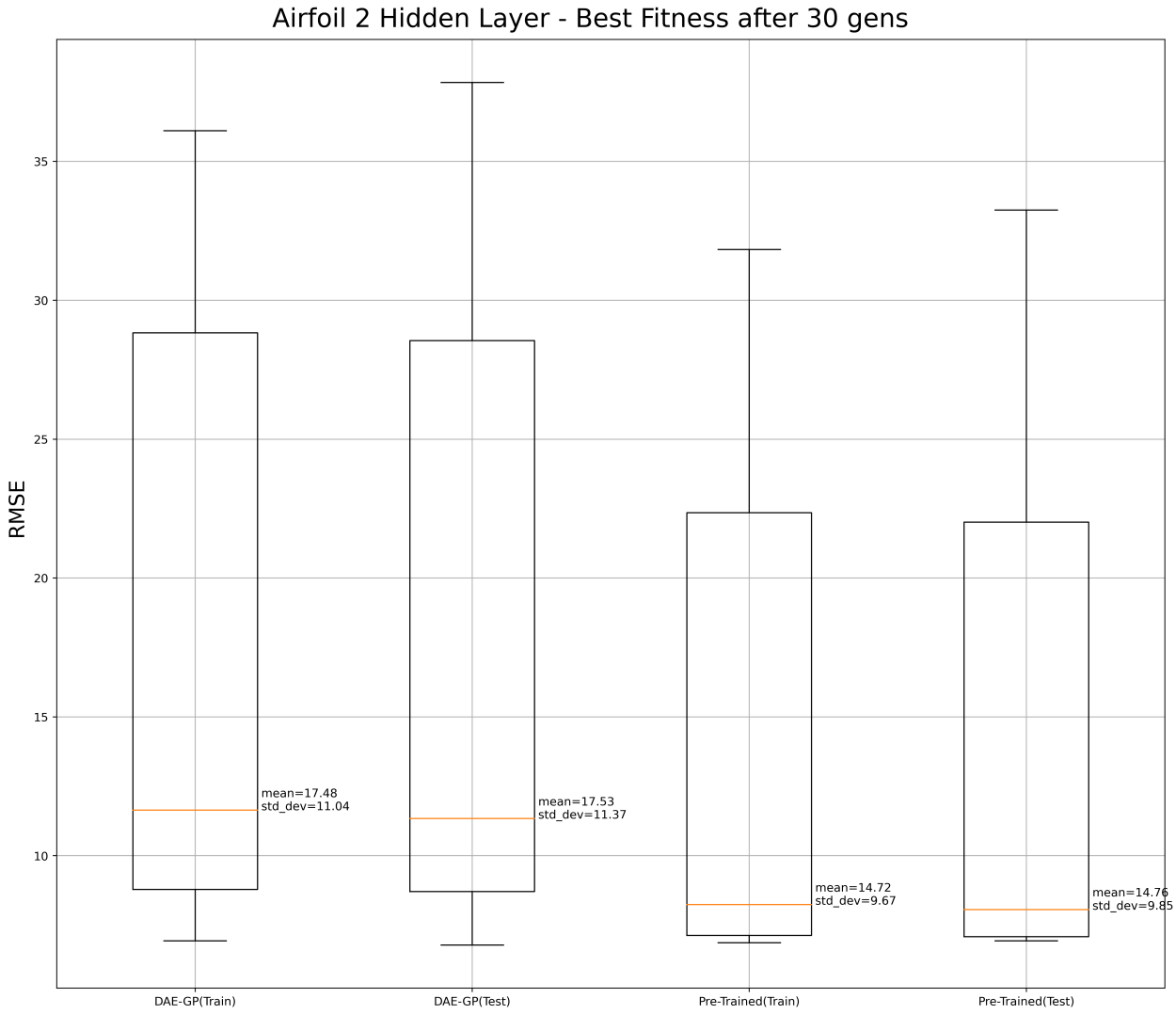
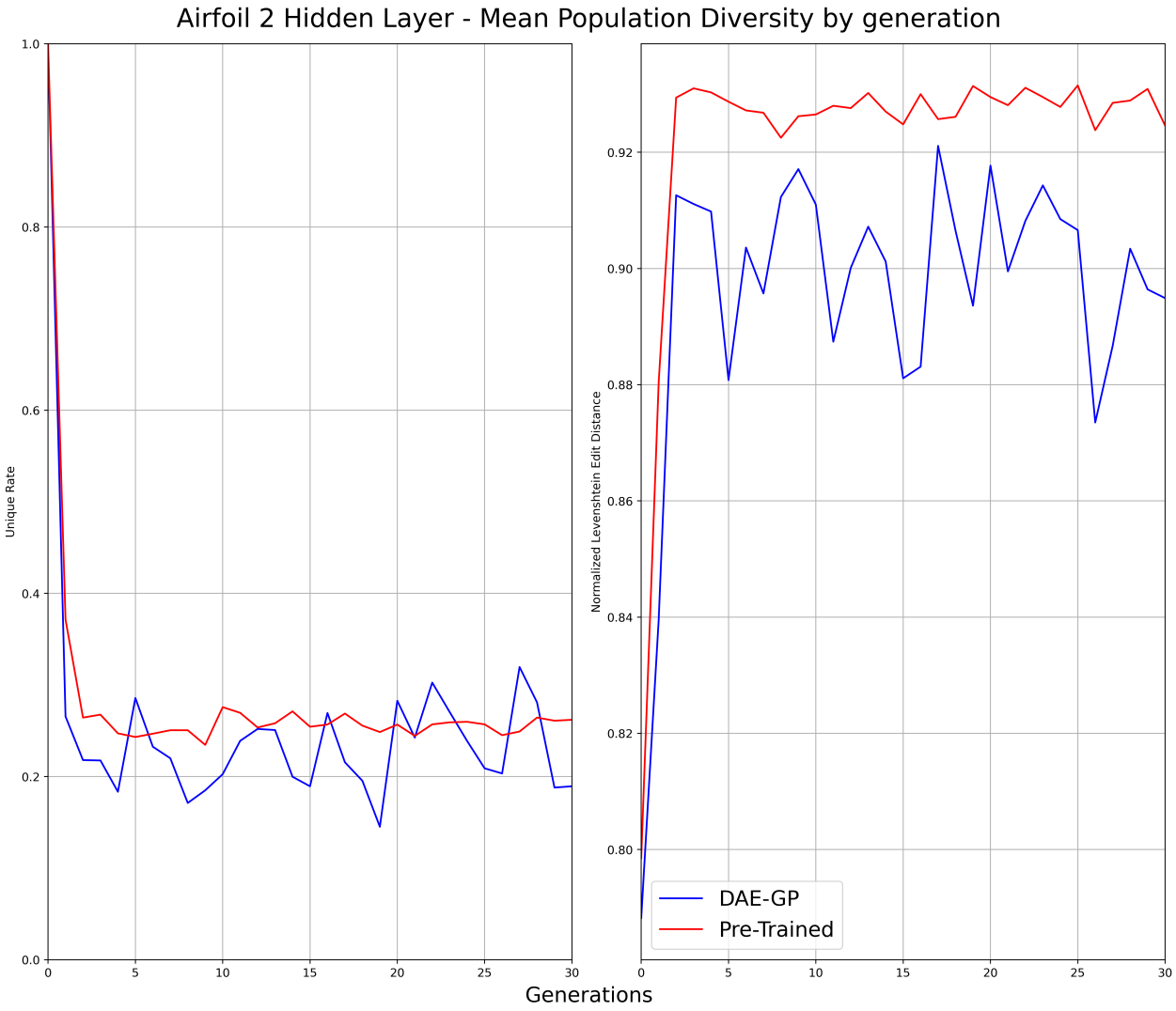



Figure 5: Best Fitness over 30 Generations - Airfoil


Airfoil 2 Hidden Layer - Best Fitness after 30 gens
RMSE
35
30
25
20
15
10
mean=17.48
std_dev=11.04
mean=17.53
std_dev=11.37
mean=14.72
std_dev=9.67
mean=14.76
std_dev=9.85
DAE-GP(Train)
DAE-GP(Test)
Pre-Trained(Train)
Pre-Trained(Test)

lalal




Since [17] shows, that DAE-GP finds higher quality that are also smaller in size than those found by average GP for a given number of 10.000 fitness evaluations, I also studied the differences in average solution size between pre-trained and regular DAE-GP.

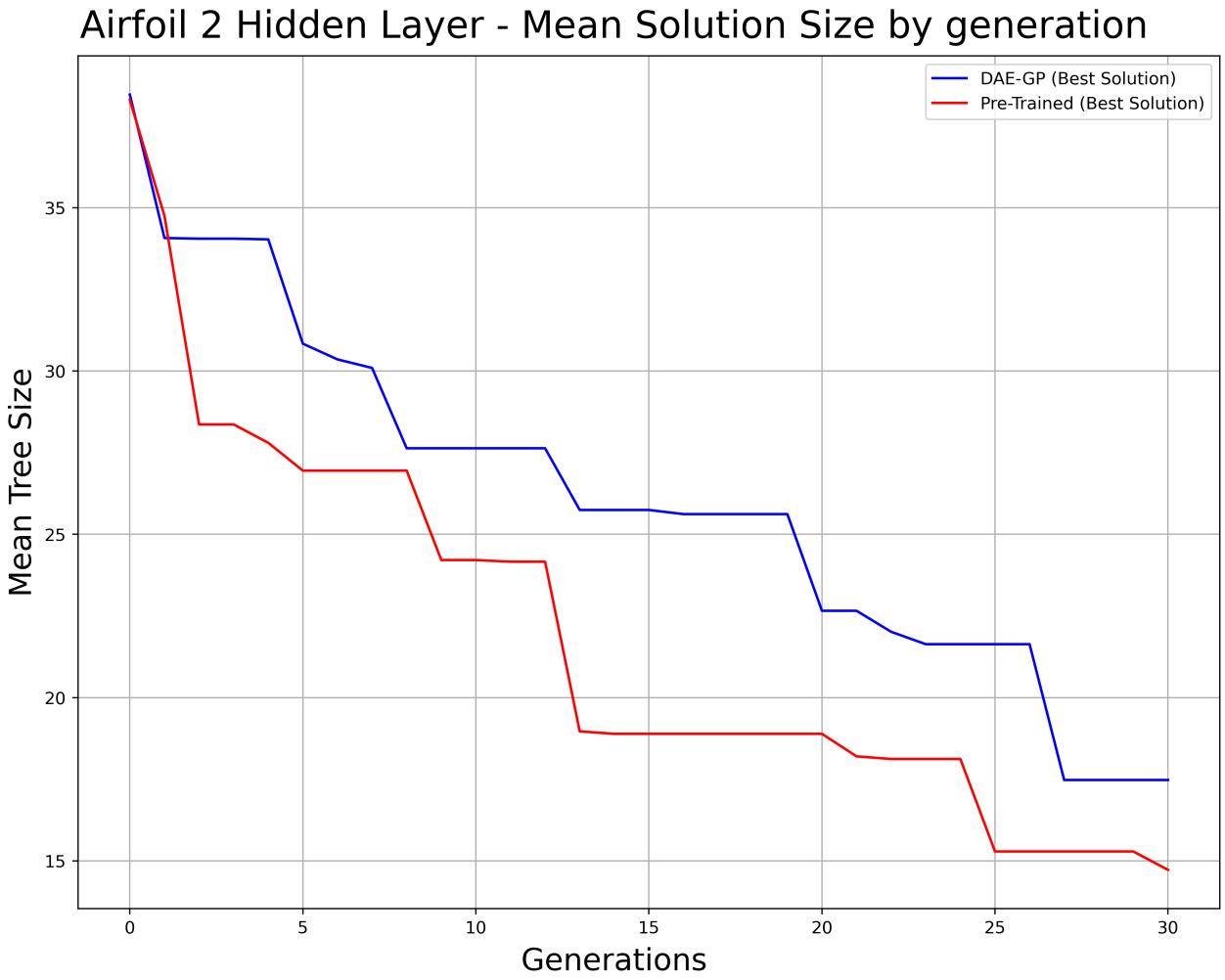



Figure 6: Average Solution Size over 30 Generations - Airfoil

The experiments was repeated using the same setup with the only difference being a reduction from two hidden layers to 1 hidden layer.

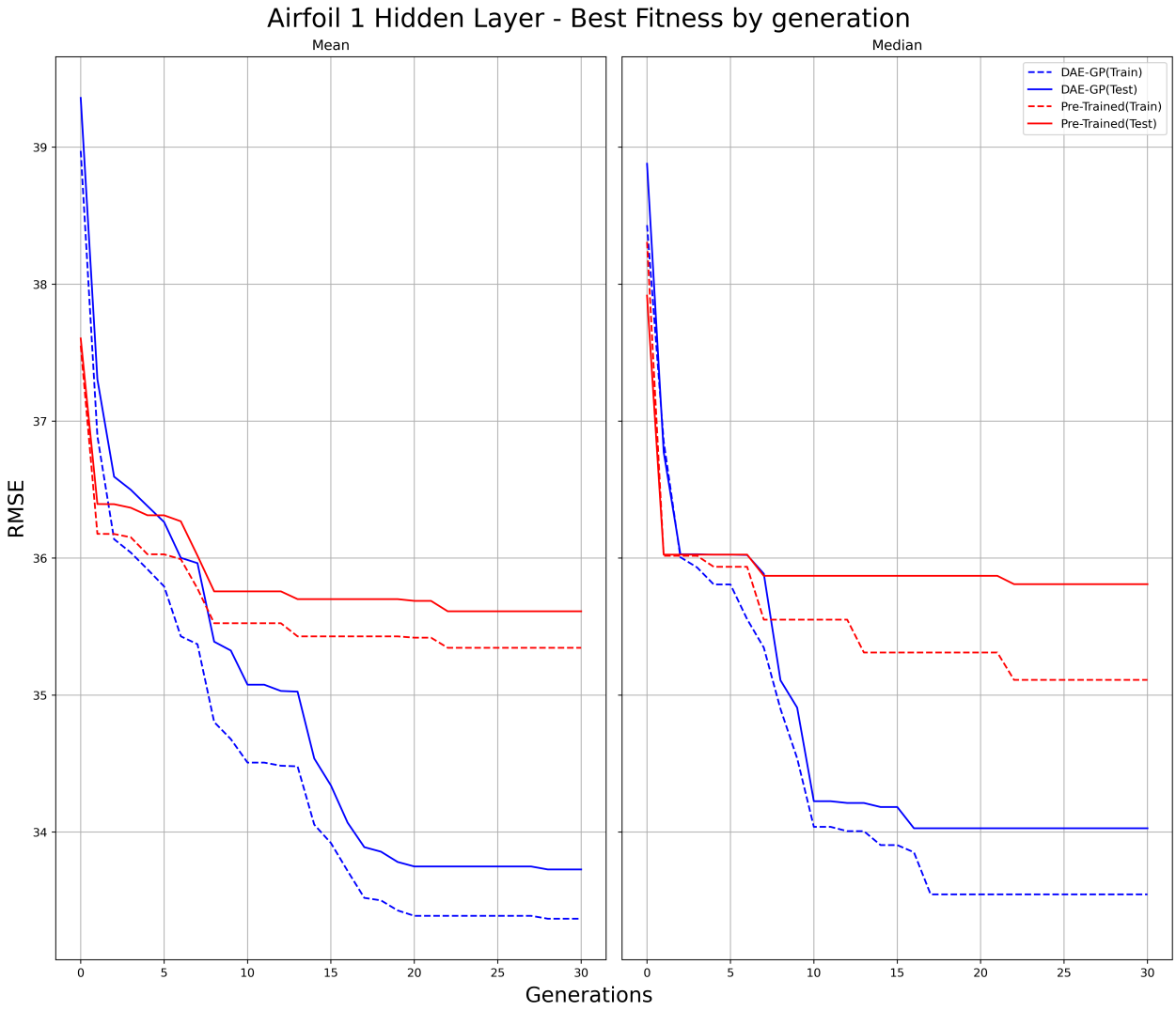



Figure 7: Best Fitness over 30 Generations - 1 hidden Layer - Airfoil

...

To gather more confidence in the previous results, the pre-training strategy for DAE-GP was tested for various real world symbolic regression problems taken from [2]. Are brief summary of the benchmark problems is given in table X

Table 4: Overview - Real World Symbolic Regression Benchmark Problems

| **Problem** | **Observations** | **Features** |
|---|---|---|
| Airfoil | 1503 | 5 |
| Boston_Housing | 506 | 13 |
| Energy_Cooling | 768 | 8 |
| Concrete | 1030 | 8 |

Table X summarizes the median best fitness achieved after running each algorithm for 10 runs

using a population size of 500 individuals for 30 generations.[1]. The median best fitness over all 30 generations are visualized in figure X.

Table 5: Median Best Fitness - Symbolic Regression

| Problem | Hidden-Layers | Set | DAE-GP | Pre-Trained | P-Value | Cliffs-Delta |
|---|---|---|---|---|---|---|
| Airfoil | 1 | Train | **33.3674** | 35.3454 | 0.02** | 0.64 |
| | 1 | Test | **33.7271** | 35.612 | 0.14 | 0.40 |
| Airfoil | 2 | Train | 17.4762 | **14.7233** | 0.31 | -0.28 |
| | 2 | Test | 17.5299 | **14.7603** | 0.57 | -0.16 |
| Boston_Housing | 2 | Train | 8.1922 | **8.0763** | 0.29 | -0.29 |
| | 2 | Test | 8.0501 | **7.9692** | 0.71 | -0.11 |
| Energy(Cooling) | 2 | Train | **4.5271** | 4.7782 | 0.02** | 0.63 |
| | 2 | Test | **4.67** | 4.9196 | 0.29 | 0.29 |
| Concrete | 2 | Train | 17.0052 | **16.9085** | 0.97 | 0.02 |
| | 2 | Test | **16.5845** | 16.9298 | 0.52 | 0.18 |

[1]P-Values are marked with asteriks (1,2,3) for ($\alpha$)-levels of 0.1, 0.05 and 0.01

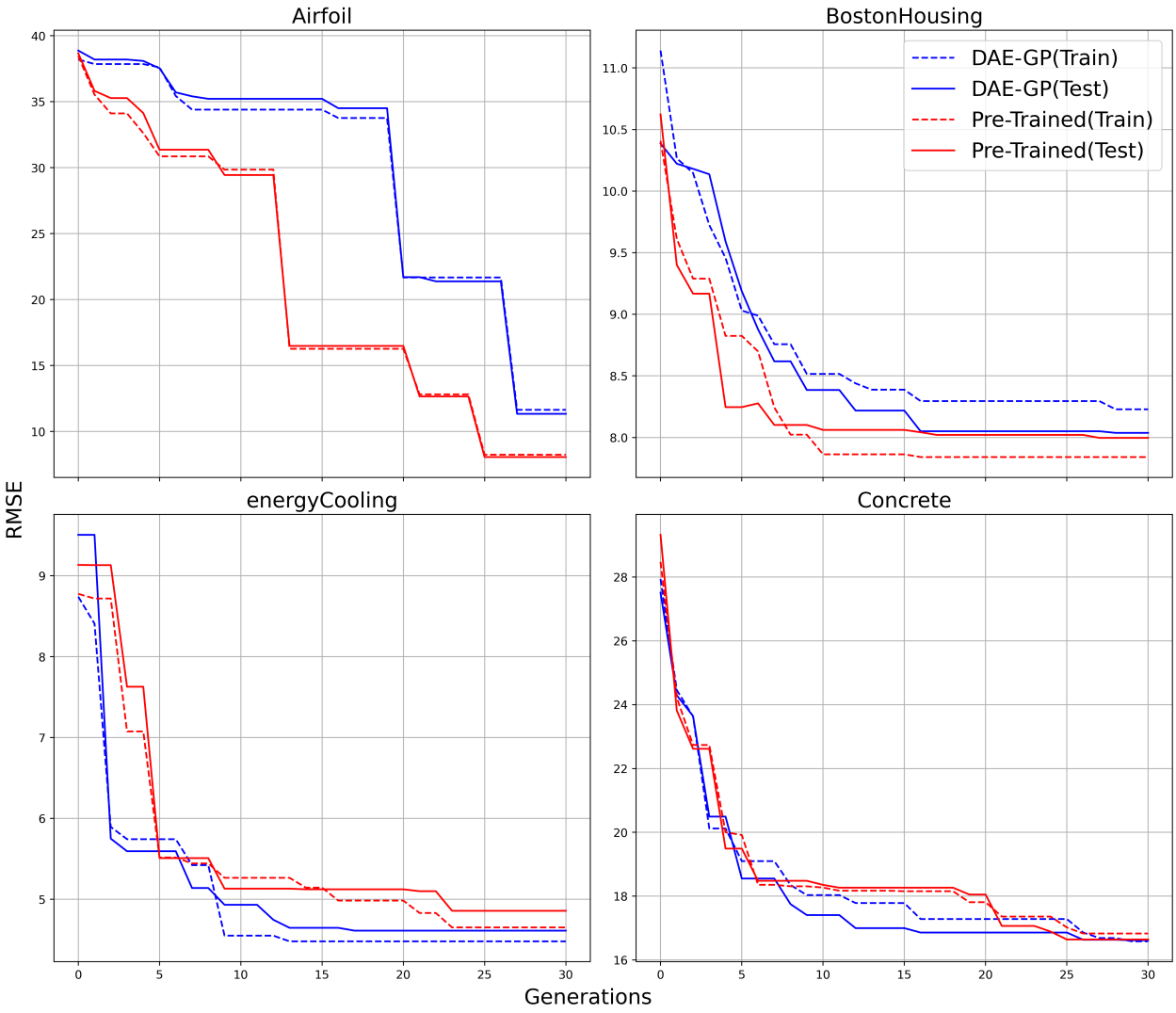



Figure 8: Median Best Fitness - Symbolic Regression

### 5.3 Solution Size in Symbolic Regression

Next I studied the size of solutions based on using a pre-training strategy. Figure X shows the median size of the best solutions aggregated of all benchmark problems for symbolic regression

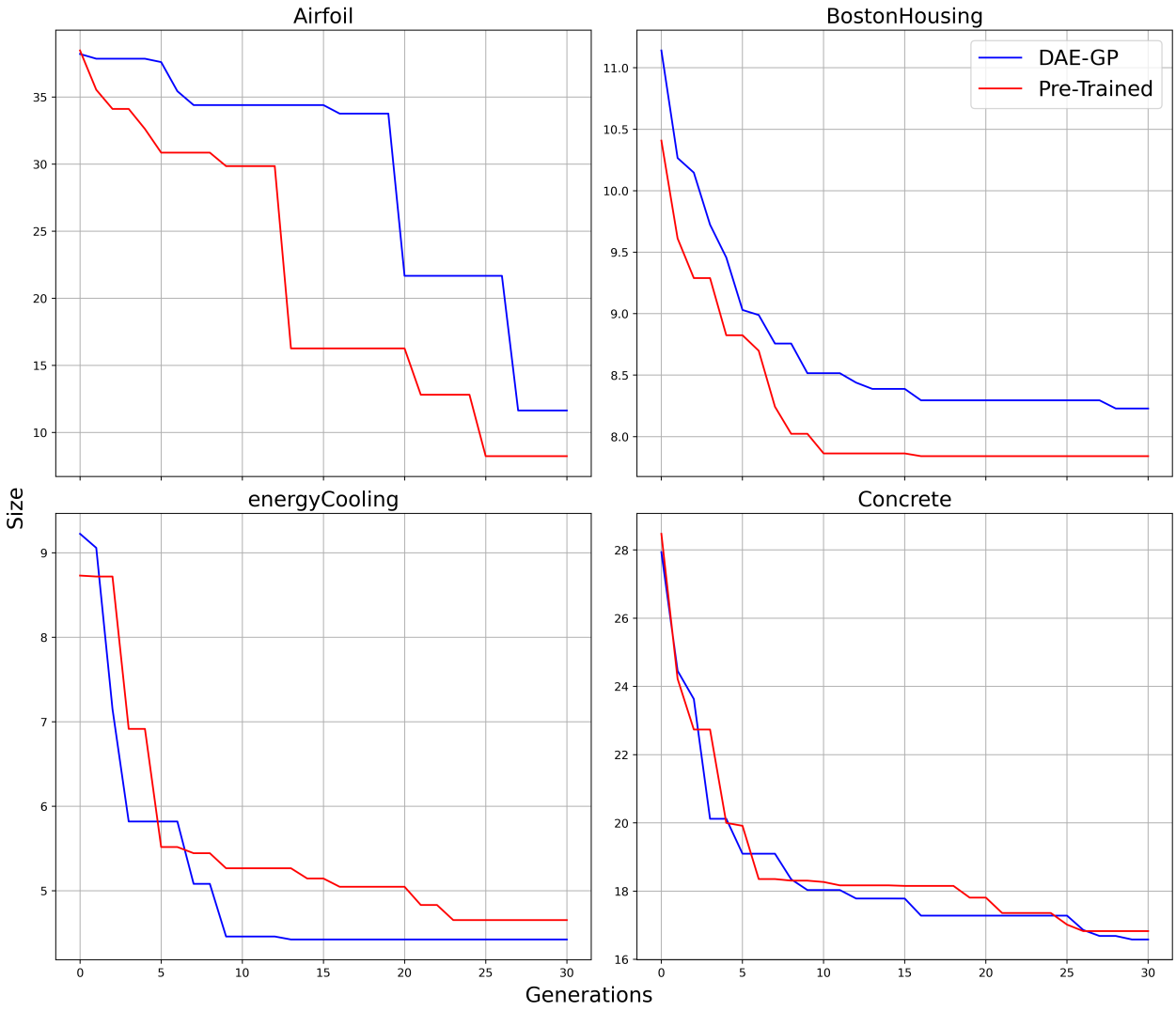



Figure 9: Median Solution Size - Symbolic Regression

Table X summarizes the results of the statistical tests.

Table 6: Median Solution Size - Symbolic Regression

| Problem | Hid.Layers | DAE-GP | Pre-Trained | P-Value | Cliffs-Delta |
|---|---|---|---|---|---|
| Airfoil | 1 | **33.3674** | 35.3454 | 0.02** | 0.64 |
| Airfoil | 2 | 17.4762 | **14.7233** | 0.31 | -0.28 |
| Boston_Housing | 2 | 8.1922 | **8.0763** | 0.29 | -0.29 |
| Energy(Cooling) | 2 | **4.5271** | 4.7782 | 0.02** | 0.63 |
| Concrete | 2 | 17.0052 | **16.9085** | 0.97 | 0.02 |

## 5.4 Population Diversity

Another interesting metric to examine in EC based metaheuristics is the diversity of the current generations population. Since the individual solutions in DAE-GP are computer programs in the

form ob parse trees, I selected the normalized Levenshtein edit distance as my primary metric for tracking population diversity. figure X shows how the median population diversity develops over the full number of generations for all benchmark problems.

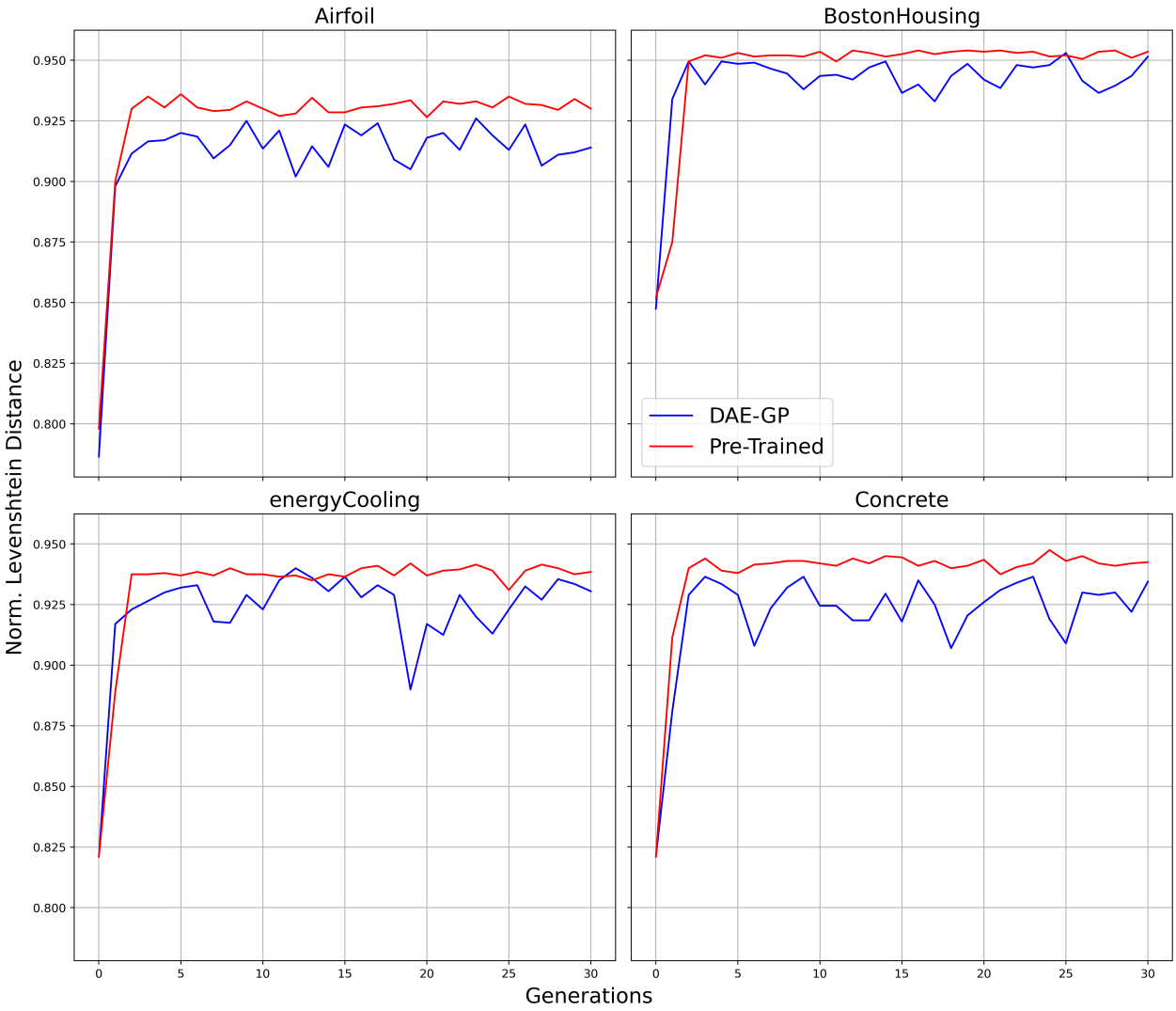



Figure 10: Median Population Diversity over Generations

Testing for statistical differences in the underlying distributions yields the P-Values listed in table X.

Table 7: Median Population Diversity over Generations - Symbolic Regression

| Problem | Hid.Layers | DAE-GP | Pre-Trained | P-Value | Cliffs-Delta |
|---|---|---|---|---|---|
| Airfoil | 1 | **0.34** | 0.49 | 0.02** | 0.36 |
| Airfoil | 2 | **0.91** | 0.93 | 0.00*** | 0.88 |
| Boston_Housing | 2 | **0.94** | 0.95 | 0.00*** | 0.82 |

| Problem | Hid.Layers | DAE-GP | Pre-Trained | P-Value | Cliffs-Delta |
|---|---|---|---|---|---|
| Energy(Cooling) | 2 | **0.93** | 0.94 | 0.00*** | 0.80 |
| Concrete | 2 | **0.93** | 0.94 | 0.00*** | 0.88 |

### 5.5 Training epochs per Generation

One important argument for using pre-training is the reduction of runtime and computational ressources for DAE-GP. An important factor in this, is the number of epochs that each DAE-LSTM has to be trained per generation. I studied the median number of training epochs that each generations DAE-LSTM $M_g$ (excluding the pre-training DAE-LSTM model $\hat{M}$) was trained until the training was stopped. The results are visualized in figure X and summarized in table X

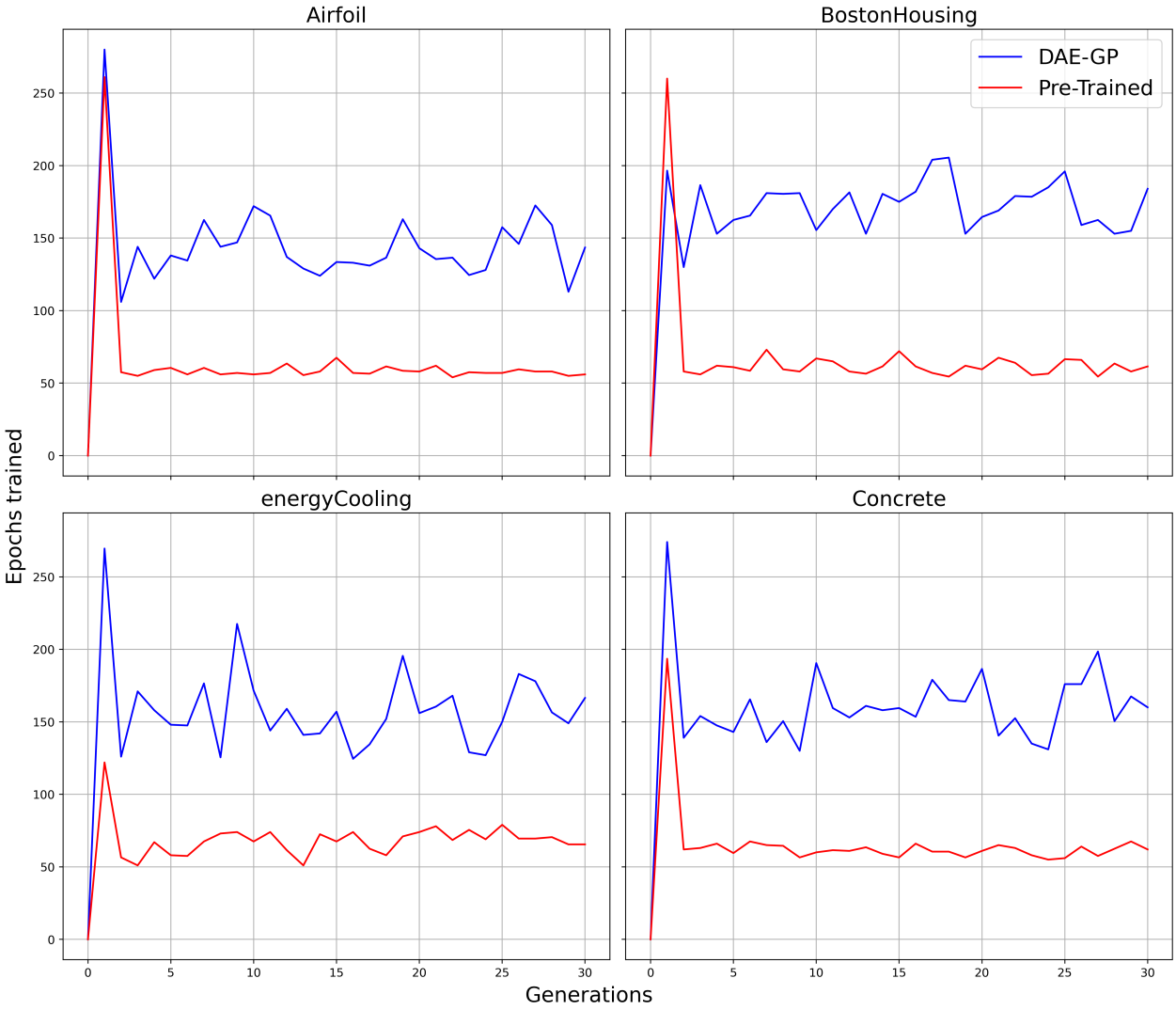



Figure 11: Median Number of Training Epochs per Generation - Symbolic Regression

Table 8: Median Number of Training Epochs per Generation - Symbolic Regression

| Problem | Hid.Layers | DAE-GP | Pre-Trained | P-Value | Cliffs-Delta |
|---|---|---|---|---|---|
| Airfoil | 1 | 190.93 | **60** | 0.00*** | -0.94 |
| Airfoil | 2 | 151.87 | **64.95** | 0.00*** | -0.88 |
| Boston_Housing | 2 | 182.84 | **67.27** | 0.00*** | -0.88 |
| Energy_Cooling | 2 | 169.39 | **69.87** | 0.00*** | -0.90 |
| concrete | 2 | 171.69 | **64.9** | 0.00*** | -0.92 |

# 6 Discussion

...

# 7 Limitations and open Questions

## 7.1 Further Questions

[3]: Pre-Training effect especially usefull for lower-level layers -> Only adapt weights from low layers or embedd the pre-trained model?

Different pre-training strategies?

## References


[1] Francois Chollet and others. 2015. Keras. Retrieved from https://github.com/fchollet/keras

[2] Dheeru Dua and Casey Graff. 2017. UCI machine learning repository. Retrieved from http://archive.ics.uci.edu/ml

[3] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. 2009. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the twelth international conference on artificial intelligence and statistics* (Proceedings of machine learning research), PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 153–160. Retrieved from https://proceedings.mlr.press/v5/erhan09a.html

[4] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13, (July 2012), 2171–2175.

[5] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *AI Open* 2, (2021), 225–250. DOI:https://doi.org/https://doi.org/10.1016/j.aiopen.2021.08.002

[6] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507. DOI:https://doi.org/10.1126/science.1127647

[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, (December 1997), 1735–80. DOI:https://doi.org/10.1162/neco.1997.9.8.1735

[8] John R. Koza. 1993. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*.

[9] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359. DOI:https://doi.org/10.1109/TKDE.2009.191

[10] Grégory Paris, Denis Robilliard, and Cyril Fonlupt. 2004. Exploring overfitting in genetic programming. In *Artificial evolution*, Springer Berlin Heidelberg, Berlin, Heidelberg, 267–277.

[11] Malte Probst and Franz Rothlauf. 2020. Harmless overfitting: Using denoising autoencoders in estimation of distribution algorithms. *Journal of Machine Learning Research* 21, 78 (2020), 1–31. Retrieved from http://jmlr.org/papers/v21/16-543.html

[12] Franz Rothlauf. 2011. *Design of modern heuristics: Principles and application*. DOI:https://doi.org/10.1007/978-3-540-72962-4

[13] Dirk Schweim, David Wittenberg, and Franz Rothlauf. 2021. On sampling error in genetic programming. *Natural computing* 2021, (2021). DOI:https://doi.org/http://doi.org/10.25358/openscience-5820

[14] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (ICML '08), Association for Computing Machinery, New York, NY, USA, 1096–1103. DOI:https://doi.org/10.1145/1390156.1390294

[15] Andreas Weigend. 1994. On overfitting and the effective number of hidden units. In *Proceedings of the 1993 connectionist models summer school*, 335–342.

[16] David Wittenberg. 2022. Using denoising autoencoder genetic programming to control exploration and exploitation in search. In *Genetic programming*, Springer International Publishing, Cham, 102–117.

[17] David Wittenberg and Franz Rothlauf. 2022. Denoising autoencoder genetic programming for real-world symbolic regression. In *Proceedings of the genetic and evolutionary computation conference companion* (GECCO '22), Association for Computing Machinery, New York, NY, USA, 612–614. DOI:https://doi.org/10.1145/3520304.3528921

[18] David Wittenberg, Franz Rothlauf, and Dirk Schweim. 2020. DAE-GP: Denoising autoencoder LSTM networks as probabilistic models in estimation of distribution genetic programming. In *Proceedings of the 2020 genetic and evolutionary computation conference* (GECCO '20), Association for Computing Machinery, New York, NY, USA, 1037–1045. DOI:https://doi.org/10.1145/3377930.3390180